• ¿Qué tipo de ejecución (compilado, interpretado, etc) tiene el lenguaje?

**FORTRAN:** Es un lenguaje de programación compilado.

PERL: Es un lenguaje de programación interpretado

• ¿Para qué tipo de desarrollo se utiliza normalmente el lenguaje?

FORTRAN: Es utilizado principalmente en matemáticas y en aplicaciones de cálculo científico.

**PERL:** Utilizado para la manipulación de cadenas de caracteres, archivos y procesos.

• ¿Con que ide o editor de texto puede utilizar el lenguaje? Nombre de una

**FORTRAN:** Para editar se puede utilizar EMACS, mientras que si necesitamos una librería podemos recurrir a ZOFU.

PERL: Uno de los frameworks más utilizados es Mojolicious, y una de sus librerías es CGI.pm

• Investigar y realizar en la sintaxis del lenguaje dado, la siguiente operación matemática:

```
x = 4
y = 5
z = x + y
mostrar por pantalla z
```

• Opcional: Crea un código que te parezca interesante o que quieras compartir con tus compañeros.

```
## Translate a simple language into HTML.

# Blank lines become
```

# stuff becomes <u>stuff</u>

# @stuff@ becomes <tt>stuff</tt>

# {stuff} becomes <i>stuff</i>

# [stuff] becomes <b>stuff</b>

# [[stuff]] becomes <h1> header.

# [:stuff:] becomes <h1> header and centered.

# A line of 3 or more --- becomes an <hr>>.

# A >>> at the start of a line becomes

# Other stuff is retained, with & < > " being translated into

# approprite stuff. The \ escapes substitutions.

# Generally, \ is removed. Use \\ to add a single one.

#

use strict;

```
# Set of plain replacements. Order is important.

my @plain = ( '\&', '&',

'^\s*\>\>\s*', '     ',

'\<', '&lt;', '\>', '&gt;', '\''', '&quot;',

'\[\[', '<h1>', '\]\]', '</h1>',

'\[\:', '<center><h1>', '\:\]', '</h1></center>',

'\\\[', '<i>', '\\]', '</b>',

'\\s*$', '',
```

```
'^\-\-+$', '<hr>' );
# Set of replacements which toggle. Replacement will be the open
# or closing version of the tag.
my @toggle = ( '\_', 'u', '\@', 'tt' );
my %toggle_status = ( '\_' => ", '\@' => " );
# Prefix.
print '<html><body bgcolor="white">', "\n";
# Read the input.
while(defined(my $line = <STDIN>)) {
  chomp $line;
  # Do the simple replacements.
  my @rs = @plain;
  while(my $r = shift @rs) {
     my $rt = shift @rs;
     }
  # Work through the toggles.
  @rs = @toggle;
  while(my $r = shift @rs) {
     my $rt = shift @rs;
     # Accumulate the new line here.
     my $newline = ";
     # Take each left-most match for $r. The *? construct
     # is a * that matches as few as possible.
     while(\frac{= /(.*?(^{[^{\]})})r(.*)$/}{
       # Move the left part to newline and add the
       # replacement, then continue with the right part.
       $newline .= "$1<$toggle_status{$r}$rt>";
       line = $3;
       # Toggle the status of the symbol.
       $toggle_status{$r} = $toggle_status{$r} ? " : '/';
     $line = $newline . $line;
  }
  # Reduce escapes.
  l =  s/\(.)/1/g;
  print "$line\n";
}
print "</body></html>\n";
```